EFREI 2010/2011

L3

Aide à la Décision

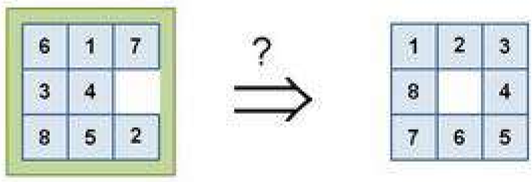
Recherche dans un espace d'états

Intelligence Artificielle
Artificial Intelligence

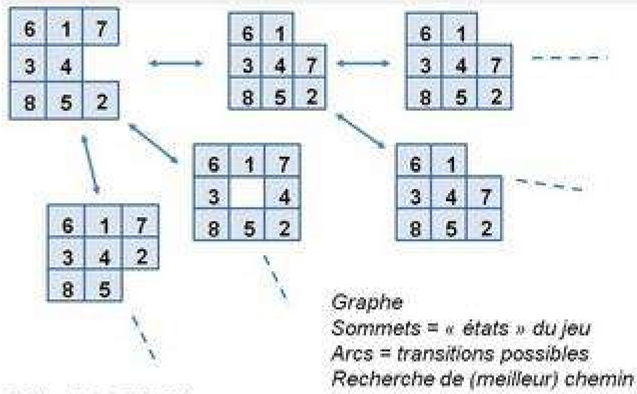
Recherche dans un espace d'états

Search in a states space

- Taquin (« 8-puzzle »)



 2



 3

- Conditions de recherche :
  - Nombre de situations (d'états) très grands
  - Représentation exhaustive de l'espace d'états irréaliste
    - Temps
    - Espace
  - Espace d'états éventuellement inconnu
    - Voire même illimité
  - Recherche de suite d'actions menant à un « but »
    « but » connu explicitement
    ou déterminé par une notion d' « acceptabilité »

 5

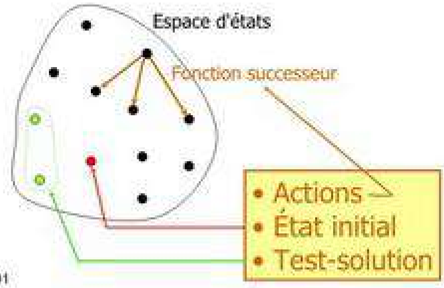
Recherche dans un espace d'états
Search in a states space

Formalisation

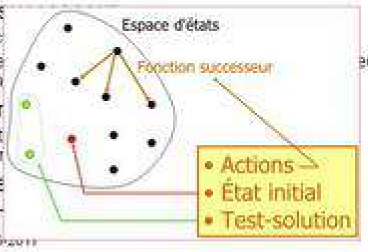
Formal description

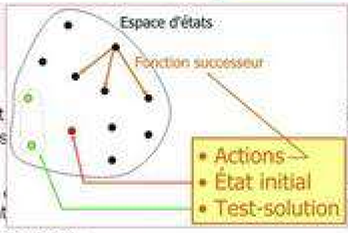
Espace d'états

- Un « état » = une situation, une étape, … un point de passage pour aller de la situation actuelle à une solution



- Espace d'états
  - Chaque état est une représentation abstraite de l'environnement
  - L'espace d'états est « discret »
- Etat initial
  - Point de départ de la recherche, généralement l'état courant de l'environnement
- Fonction « s
  - état x ac
  - Représe … eurs effets sur l'environ
- Test solution
  - Descript
  - condition
- Coût du che
  - chemin -





- Espace d'états
  - Chaque état est … ment
  - L'espace d'états
- Etat initial
  - Point de départ … t de l'environnement
- Fonction « successeur »
  - état x action → état
  - Représentation abstraite des actions possibles et de leurs effets sur l'environnement
- Test solution
  - Description explicite d'un état, ou condition à satisfaire
- Coût du chemin
  - chemin → nombre (positif ?)

Paramètres d'un problème de recherche

- Q = { états }
  Non vide – Supposé fini (en général)
- I = { états initiaux } … *ou « { état initial } »*
  $I \subseteq Q$
- T = { états « solution » }
  $T \subseteq Q$
  Connus explicitement ou déterminés implicitement par une fonction test-solution
- A = { actions }
- $S : Q \times A \rightarrow Q$
  Passage d'un état à un autre au moyen d'une certaine action
  Notation : $S(q) = \{ q_i \in Q \mid \exists a_i \in A, S(q,a_i) = q_i \}$

- Une solution à un problème =
  - une suite d'actions $a_0, a_1, \ldots, a_{n-1}$
  - une suite d'états $q_0, q_1, \ldots, q_n$
  telle que
  $\forall i, 0 \le i \le n, q_i \in Q$
  $\forall i, 0 \le i < n, a_i \in A, q_{i+1} = S(q_i,a_i)$
  $q_0 \in I$
  $q_n \in T$

- S^{-1} : Q x A → Q fonction « prédécesseurs »

- S^* : Q x A* → Q fonction « descendants »

  A* = ensemble / suite d'actions

(C) Hervé Barbot, 2005-2011 14

- C : Q x A [x Q] → $\Re^+$
  C : Q x Q → $\Re^+$

  Coût d'une transition
  défini en relation avec la fonction S

- C* : Q x A* [x Q] → $\Re^+$
  C* : Q* → $\Re^+$

  Coût d'un chemin défini par :
  $\forall i,\ 0 \le i < n,\ q_i \in Q,\ q_{i+1} = S(q_i, a_i)$
  $C^*(q_0, q_1, \ldots, q_n) = \sum_{0 \le i < n} C(q_i, q_{i+1})$

(C) Hervé Barbot, 2005-2011 15

Enoncé d'un problème

- Etant donné une situation donnée q_0
  - L'état actuel
  - Le point de départ de la recherche
- Comment faire pour aboutir à une situation q_n acceptable ?
  - Quelles actions effectuer pour atteindre un état solution ?
    - Comment le faire à « moindre coût » ?

Quel chemin (q_0, q_1, …, q_n) de plus faible coût ?
(Minimisation de la fonction C*)

(C) Hervé Barbot, 2005-2011 16

Stratégies de recherche

- Stratégies aveugles
  - Largeur d'abord
  - Profondeur d'abord
  - Coût uniforme
  - Bi-directionnelle

Aucune information particulière n'est contenue dans un nœud donné

- Stratégies heuristiques
  - « best-first »
  - Recherche gourmande
  - A*

Des informations existent pour déterminer si un nœud est « plus intéressant » qu'un autre

(C) Hervé Barbot, 2005-2011 17

Critères d'évaluation des algorithmes

- Complétude
  - La méthode garantit-elle de trouver une solution si elle existe.
- Optimalité
  - Si plusieurs solutions existent, la méthode garantit-elle de trouver la « meilleure » ?
- Complexité en temps
  - Combien de nœuds (d'états) faut-il produire / analyser pour trouver la solution ? *(ordre de grandeur)*
- Complexité en espace
  - Combien de nœuds faut-il conserver en mémoire pour trouver une solution ? *(ordre de grandeur)*

(C) Hervé Barbot, 2005-2011 18

- Parameters to be used for numeric values:
  - b = (maximum) number of successors for a given state
  - d = depth of the solution that is found when applyng a specific strategy
  - m = maximum depth of the search tree

(C) Hervé Barbot, 2005-2011 19

Exploration d'arbres

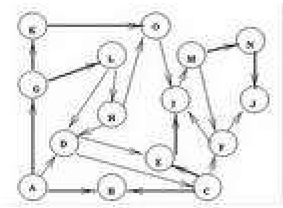
Exploring trees

(C) Hervé Barbot, 2005-2011 20

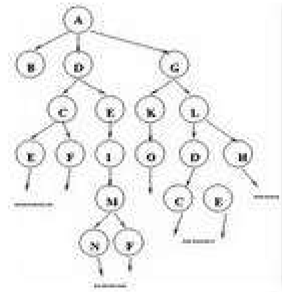
Représentation sous forme d'arbre

- Arbre « de dérivation » ou « de recherche »

Problème général



Représentation par arbre



(C) Hervé Barbot, 2005-2011 21

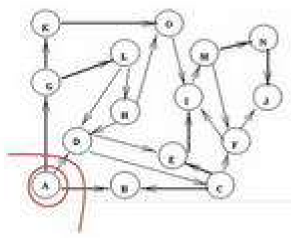
- Arbre « de dérivation » ou « de recherche »
  - A chaque nœud : ses successeurs
  - Certains nœuds représentent des solutions
  - Choisir un successeur = se déplacer dans l'arbre
    - Selon la stratégie choisie
- Etat
  - Configuration réelle de l'environnement, du problème à résoudre
- Nœud
  - Élément de la SdD qui représente un état
  - Notions de parent / fils, de profondeur, de coût d'un chemin

(C) Hervé Barbot, 2005-2011 23

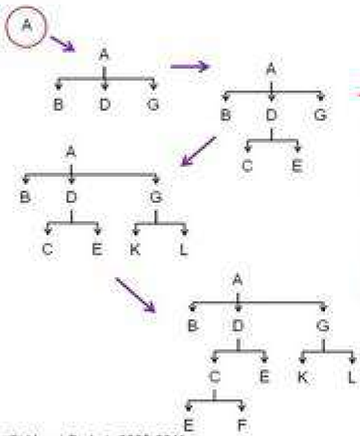
Exploration off-line simulée

A

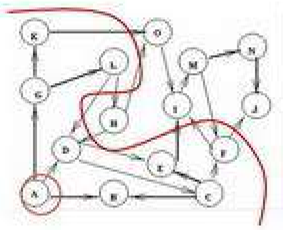
Problème général



(C) Hervé Barbot, 2005-2011 24



After a few steps:



The left-hand side of the line is the part of the states space that has been considered

(C) Hervé Barbot, 2005-2011 25

- Gestion d'un ensemble d'états « candidats à expansion »
  - Génération des successeurs des états déjà explorés
  - Stratégie de choix du successeur pour la progression de la recherche
  - Atteinte d'une solution

(C) Hervé Barbot, 2005-2011 26

Recherche « offline » - Algorithme

- Exploration simulée de l'espace d'états
- Processus d'expansion
  - Génération des états successeurs de ceux explorés

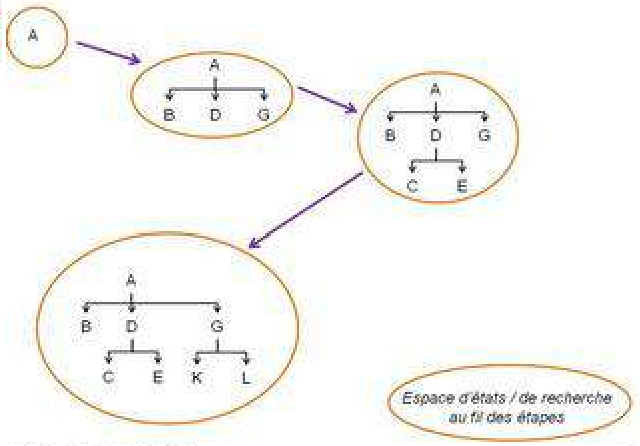
function Tree-Search (*problem, strategy*) **returns** a solution, or failure
  initialize the search tree using the initial state of *problem*
  loop do
    if there are no candidates for expansion **then return** failure
    choose a leaf node for expansion according to *strategy*
    if the node contains a goal state **then return** the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end

Recherches aveugles

Blind search

Starting point – rev 0

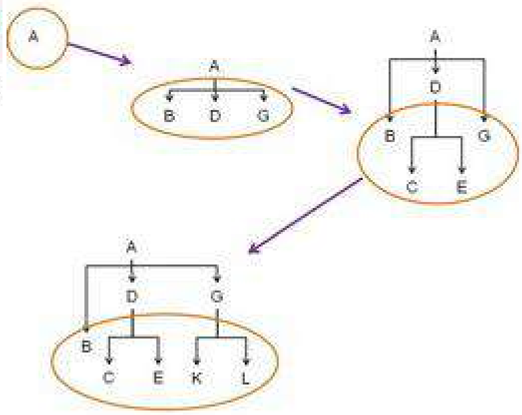
```
states ← {initial state} // States to explore
succes ← false

Loop while ¬ succes do

  state e ← select_according_to_strategy_in(states)

  is_final(e) ⇒ succes ← true
  ¬ is_final(e) ⇒ states ← states + S(e)
```

Rev 1

- Un état comparé avec la solution n'a plus à être considéré

→ Enlever les états « choisis » de l'espace de recherche



Rev 2

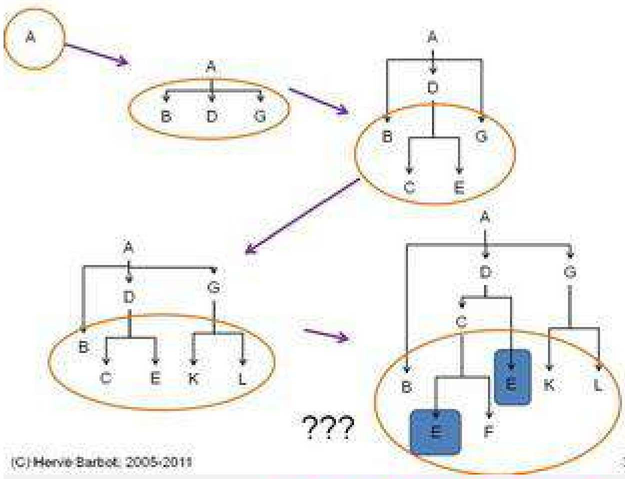
- Et s'il n'y avait pas d'état « solution » ?...
  - Problème insoluble depuis l'état initial…

→Doit entraîner un parcours complet de l'espace d'états (i.e. de l'arbre de recherche)
avec arrêt lorsque tout celui-ci a été considéré

(C) Hervé Barbot, 2005-2011 34

```
states ← {initial state} // States to explore
succes ← false

Loop while ¬ succes ∧ states ≠ ∅ do

  state e ← select_according_to_strategy_in(states)

  is_final(e) ⇒ succes ← true
  ¬ is_final(e) ⇒ states ← states - {e} + S(e)
```

(C) Hervé Barbot, 2005-2011 35



(C) Hervé Barbot, 2005-2011 36

Rev 3

- Et si on rencontre un état qui a déjà été « rencontré » mais pas « choisi » ?

→ Ne le mettre qu'une seule fois dans l'ensemble des états à traiter !

(C) Hervé Barbot, 2005-2011 37

Autres considérations…

- Un état rencontré et choisi ne doit pas être sélectionné à nouveau
  → Sauvegarde des états déjà « choisis »
    →Ensembles 'ouverts' et 'fermés'
- Connaissance de la suite d'actions à effectuer depuis l'état initial
  → Sauvegarde du chemin parcouru de l'état initial à un état rencontré
    → Sauvegarde « à rebours » par la connaissance de l'antécédent direct
      → fonction / variable 'prédécesseur'

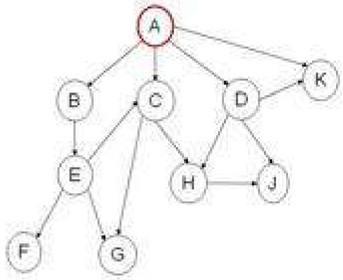
(C) Hervé Barbot, 2005-2011 39

```
opened ← { initial states } // States to be explored
closed ← ∅  // States already selected
                // and compared to solution
success ← false

Loop while opened ≠ ∅ ∧ ¬ success do
  state e ← select_according_to_strategy_in( opened )
  is_final(e) ⇒ success ← true
  ¬ is_final(e) ⇒ opened ← opened - {e}
                  closed ← closed + {e}
                  ∀ x ∈ S(e),
                     x ∉ opened ∪ closed ⇒
                              opened ← opened + {x}
                              S⁻¹(x) ← e
```

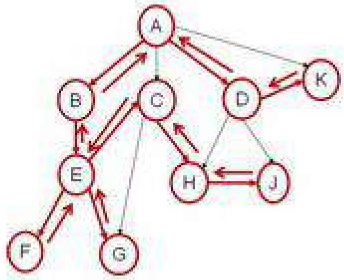
(C) Hervé Barbot, 2005-2011 40

Recherche en profondeur d'abord

- Stratégie : s'éloigner au maximum de l'état initial



(C) Hervé Barbot, 2005-2011 41



(C) Hervé Barbot, 2005-2011 42

- Stratégie : s'éloigner au maximum de l'état initial
  - Éloignement calculé en nombre d'actions, i.e. en nombre d'arcs dans l'arbre

- Implémentation : pile (Last In First Out)
  - Dans « `choix_selon_stratégie_dans` »

(C) Hervé Barbot, 2005-2011 43

- Alternative de mise en œuvre

→ Profondeur d'abord = Algorithme récursif

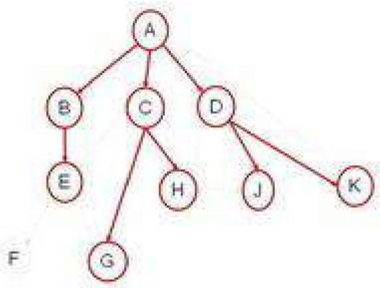
- L'ajout dans « ouverts » des successeurs de l' « état choisi » est remplacé par un appel récursif sur chaque sous-arbre
- La procédure récursive rend « vrai » si un état solution est trouvé
- Si un successeur provoque le retour « vrai », on remonte immédiatement le résultat

(C) Hervé Barbot, 2005-2011 45

Recherche en profondeur limitée

- Stratégie : recherche en profondeur d'abord avec limite L de profondeur d'analyse de l'arbre

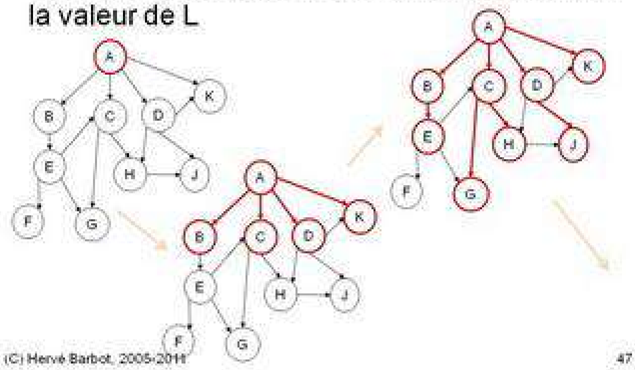
Ex. : L=2



(C) Hervé Barbot, 2005-2011 46

Recherche par approfondissement itératif

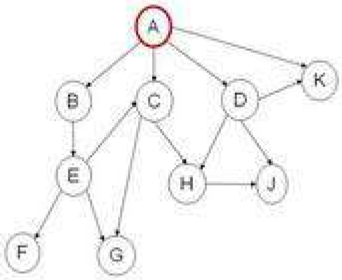
- Stratégie : profondeur limitée avec itération sur la valeur de L



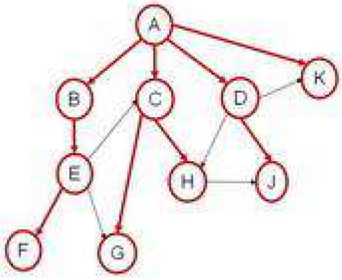
(C) Hervé Barbot, 2005-2011 47

Recherche en largeur d'abord

- Stratégie : tout vérifier au plus près de l'état initial



(C) Hervé Barbot, 2005-2011 48



(C) Hervé Barbot, 2005-2011 49

- Stratégie : tout vérifier au plus près de l'état initial
  - c.a.d. étendre le nœud le moins profond

- Implémentation : file (FIFO)

(C) Hervé Barbot, 2005-2011 50

Calcul du coût cumulé minimal 'g'

- Notion de coût
  - $C : Q \times A \times Q \rightarrow \Re^+$

    Coût défini en relation avec la fonction S
  - $C^* : Q \times A^* \times Q \rightarrow \Re^+$

    Coût d'un chemin

    $\forall i,\ 0 \le i \le n\text{-}1,\ q_i \in Q,\ q_{i+1} \in S(q_i)$

    $C^* (q_0, q_1, \ldots, q_n) = \sum_{0 \le i \le n-1} C(q_i, q_{i+1})$

- Coût minimal
  - Calculé sur l'ensemble des chemins détectés de q_0 à q_n

- **g** : $Q \rightarrow \Re^+$
  - g (e) = min C* (état initial de la recherche, …, e)
  - g (état initial) = 0
  - $x \in S(e) \Rightarrow g(x) = g(e) + C(e,s)$

(C) Hervé Barbot, 2005-2011 52

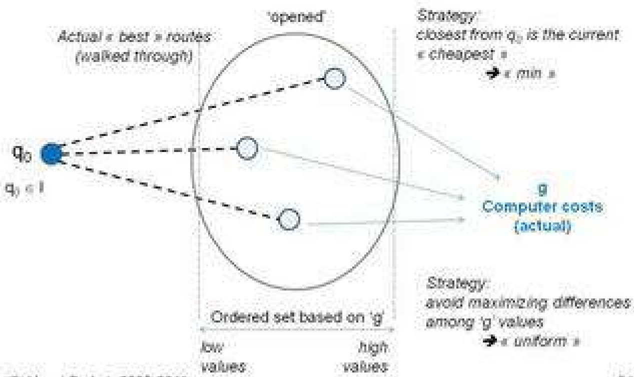
Recherche en coût cumulé (g) uniforme

- Stratégie : recherche en « pseudo-largeur » avec extension systématique du nœud de coût le plus faible

  « coût (cumulé) uniforme » =
  - On ne se lance pas dans une branche de l'arbre de recherche pour laquelle on a déjà un coût cumulé supérieur aux autres branches
  - On regarde en priorité les branches 'ouvertes' de coût moindre
    - C-à-d qu'on limite la plus grande différence de coût cumulé entre deux états de l'ensemble 'ouverts'

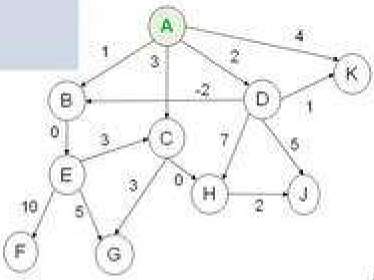
(C) Hervé Barbot, 2005-2011 53

Uniform cost



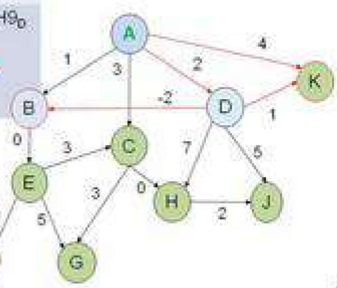
(C) Hervé Barbot, 2005-2011 54

Fermés Closed	Ouverts Opened
	A0

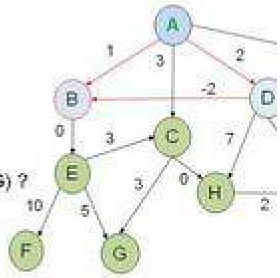


(C) Hervé Barbot, 2005-2011 55

Fermés Closed	Ouverts Opened
	A0
A0	$B1_A$ $C3_A$ $D2_A$ $K4_A$
A0 $B1_A$	$C3_A$ $D2_A$ $K4_A$ $E1_B$
A0 $B1_A$ $E1_B$	$C3_A$ $D2_A$ $K4_A$ $F11_E$ $G6_E$
A0 $E1_B$ $D2_A$	$C3_A$ $F11_E$ $G6_E$ $H9_D$ $J7_D$
	$K4_A$ $K3_D$???
$B1_A$	**$B0_D$???**



(C) Hervé Barbot, 2005-2011 56



- g(B) ← 0 ?
  - OK, mais que devient g(E) ?
  - Et g(E) ← 0 ?
    - Mais que deviennent g(F) et g(G) ?
    - …etc.
- « backtracking »
  → Remettre 'B' dans 'ouverts'
  - Il sera à nouveau « choisi » puis g(E) modifié et remis dans 'ouverts'
    - Puis g(F) et g(G) modifiés avec F et G remis dans 'ouverts'
      - …etc
- Mais : perte d'optimalité ! (ex. B et D solutions)

(C) Hervé Barbot, 2005-2011 57

```
opened ← ( initial states )
closed ← ∅
succes ← false
Loop while opened ≠ ∅ ∧ ¬ succes do
  state e ← select_according_to_uniform_cost_in ( opened )
  is_final(e)  ⇒ success ← true
  ¬ is_final(e)  ⇒ opened ← opened - e
                   closed ← closed + e
                   ∀ x ∈ S(e),
                     x ∉ opened ∪ closed
                           ⇒ opened ← opened+{x}
                             g(x) ← g(e) + C(e,s)
                             S⁻¹(x) ← e
                     x ∈ opened ∧ g(x) < g(e)+C(e,s)
                           ⇒ g(x) ← g(e)+C(e,s)
                             S⁻¹(x) ← e
                     x ∈ closed ∧ g(x) < g(e)+C(e,s)
                           ⇒ g(x) ← g(e)+C(e,s)
                             S⁻¹(x) ← e
                             closed ← closed - {x}
                             opened ← opened + {x}
```

(C) Hervé Barbot, 2005-2011 58

Recherche heuristique

Heuristic search

(C) Hervé Barbot, 2005-2011 59

Stratégies de recherche

- Recherche aveugle :
  - Aucune information sur la structure de l'arbre de recherche
  - « rustique » / systématique
- Recherche heuristique :
  - Information disponible
  - Amélioration du processus de recherche

(C) Hervé Barbot, 2005-2011 60

Fonction heuristique

- h : Q → $\mathcal{R}$
  - Associe une valeur à chaque état
    - La valeur de h(e) est indépendante des circonstances dans lesquelles on « rencontre » l'état 'e'
- h(e) = estimation du rapport coût / bénéfice
  - If we reach a solution state moving through 'e'
- Propriété fondamentale :

  h (solution state) = 0

- Soit h* (e) le coût réel minimum de 'e' à un état solution
  - h* est une valeur qui n'est pas calculée !
  - h* est inconnue !!
- h (e) – h* (e) évalue la qualité de la fonction heuristique h choisie
  - La performance d'un algorithme de recherche heuristique dépendra de la fonction h choisie !

- Fonction heuristique « monotone » ou « consistante » :

  $h(s) < h(e) + C(e\text{-->}s)$

  $e,s \in Q$, $s \in S(e)$
- Fonction heuristique « presque parfaite » :

  h établit une relation d'ordre identique à celle définie par h*

  L'ordre de choix des états dans l'ensemble 'ouverts' sera identique à celui correspondant à une mise en œuvre théorique de h* !

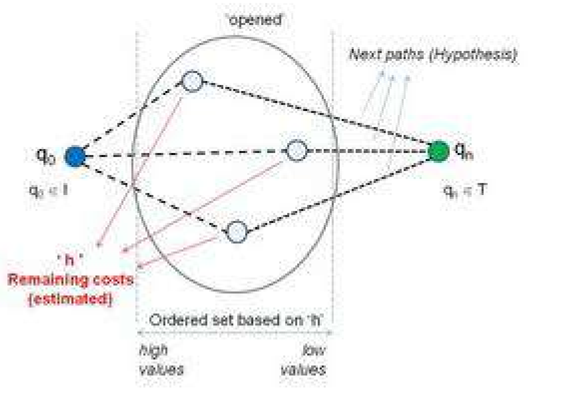
Recherche gloutonne / gourmande

« greedy search »

- Principe : à chaque étape, choisir le successeur ayant le « coût restant estimé » minimum

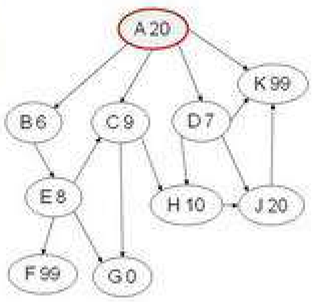
  Parmi l'ensemble des états candidats à expansion $s_1, s_2, \ldots, s_n$, on prend un état s tel que

  $h(s) = MIN(h(s_i))$

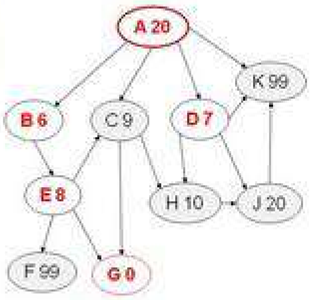


Greedy search strategy: select the lowest values of 'h'

Closed	Opened
	A

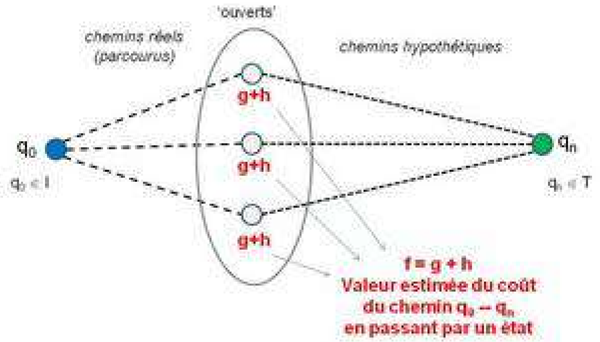


Closed	Opened
	A
A	B C D K
A B	C D K E
A B D	C K E H J
A B D E	C K H J F G
A B D E G	



(C) Hervé Barbot, 2005-2011 68

A *

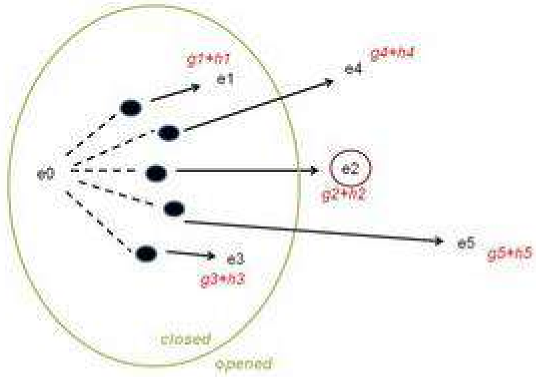


(C) Hervé Barbot, 2005-2011 69

- Stratégie A* : parmi les états candidats à expansion, choisir un de ceux qui minimise h+g
- Propriétés :
  - Complétude : OUI
  - Optimalité : OUI
  - Complexité en temps : exponentielle
  - Complexité en espace : garde tous les nœuds analysés en mémoire

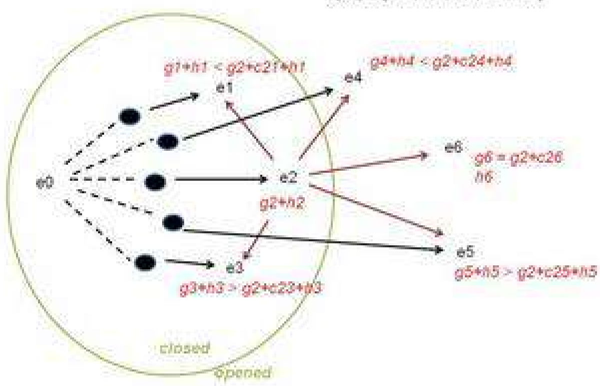
(C) Hervé Barbot, 2005-2011 70

Example



(C) Hervé Barbot, 2005-2011 71

S(e2) = { e1 , e4 , e6 , e5 , e3 }



(C) Hervé Barbot, 2005-2011 72

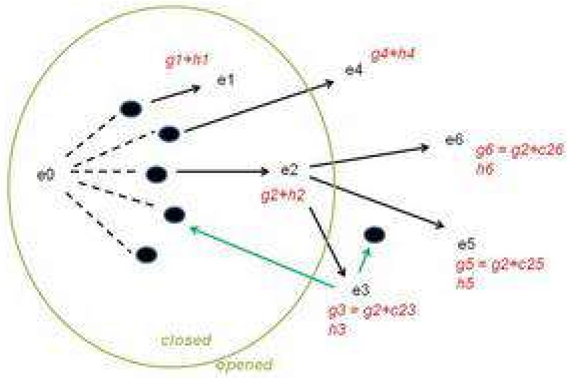
Remise en cause

- Problème : que se passe-t-il pour les descendants 'x' (successeurs directs et indirects) de e3 ?
  - $g3+h3 > g2 + c23 + h3$
    $g3 > g2 + c23$
    donc g3 va diminuer !
  - gx = g3 + « quelque chose » $x \in S(e3)$
    gx doit logiquement diminuer

(C) Hervé Barbot, 2005-2011 73

Remise en cause

- Solutions :
  - On analyse immédiatement toute la descendance de e3
    - Mais comment faire si on a déjà exploré un sous-arbre relativement important « sous » e3 ?
      - D'autant plus que la seule information liée à la structure de l'arbre est la fonction « prédécesseur »…
  - On laisse A* fonctionner tout seul pour le faire



(C) Hervé Barbot, 2005-2011 74

(C) Hervé Barbot, 2005-2011 75

```
opened ← { initial state }
closed ← ∅
succes ← false

Loop while opened ≠ ∅ and ¬ succes do
  state e ← select_with_min_f_in ( opened )
  is_final(e) ⇒ succes ← true
  ¬ is_final(e) ⇒ opened ← opened - e
                  closed ← closed + e
                  ∀ x ∈ S(e),
                    x ∉ opened ∪ closed
                      ⇒ opened ← opened+{x}
                         g(x) ← g(e)+C(e,s)
                         S⁻¹(x) ← e
                    x ∈ opened ∧ f(x) > g(e)+C(e,x)+h(x)
                      ⇒ g(x) ← g(e)+C(e,s)
                         S⁻¹(x) ← e
                    x ∈ closed ∧ f(x) > g(e)+C(e,x)+h(x)
                      ⇒ g(x) ← g(e)+C(e,s)
                         S⁻¹(x) ← e
                         closed ← closed - {x}
                         opened ← opened - {x}
```

(C) Hervé Barbot, 2005-2011 76

Adapting A* implementation

- Let h(e) = 0 for each state e
- → f = g
- 'f' min = 'g' min

Uniform cost

- Let g(e) = 0 for each state e
- → f = h
- 'f' min = 'h' min

Greedy search

(C) Hervé Barbot, 2005-2011 77

- Let h(e) = g(e) = 0 for each state 'e'

→ f = 0

- 'f' min = ???

(C) Hervé Barbot, 2005-2011 78